HW 12 Key

turn in an empty text file to signify you are finished

This homework very much based on the example at
http://stackoverflow.com/questions/37026/java-notify-vs-notifyall-all-over-again

It contains a detailed explanation of what happens -- look at it and the solution after doing this yourself
• In this homework you will show the status of the program as \textit{wait}, \textit{notify} and \textit{notifyAll} calls are made.

• This homework will demonstrate how threads wait on locks, how \textit{wait}, \textit{notify} and \textit{notifyAll} work, and why you need a \textit{while} loop to recheck a condition.
wait, notify and notifyAll

Let \( L \) be the lock

**notify**

If one or more threads are in the *wait queue*, wake one up and place it into the *blocked queue*. The thread to be woken up is picked arbitrarily. The thread woken up must acquire \( L \) before continuing. The thread executing *notify* must hold the lock \( L \) and continues to hold it until it reaches the end of the synchronized block.

**notifyAll**

If one or more threads are in the *wait queue*, wake all of them up. All woken up threads will be placed into the *blocked queue* and attempt to acquire \( L \) when it is released by the thread executing the *notifyAll*. At most one will get the lock, all others will continue to be in the *blocked queue* (not the *wait queue*!) The thread executing *notifyAll* must hold the lock \( L \) and continues to hold it until it reaches the end of the synchronized block.

**Wait**

Put the thread executing *wait* into \( L \)’s wait queue. The thread executing *wait* must hold \( L \) and releases it when it executes *wait*. 
First scenario -- this code is part of a class that implements a blocking queue

```java
public synchronized void put(Object o) {
    while (buf.size()==MAX_SIZE) {
        wait(); // called if the buffer is full (try/catch removed
        // for brevity)
    }
    buf.add(o);
    notify(); // called in case there are any getters or putters waiting
}

public synchronized Object get() {
    // Y: this is where C2 tries to acquire the lock (i.e. at the
    // beginning of the method)
    while (buf.size()==0) {
        wait(); // called if the buffer is empty (try/catch removed
        // for brevity)
        // X: this is where C1 tries to re-acquire the lock (see below)
    }
    Object o = buf.remove(0);
    notify(); // called if there are any getters or putters waiting
    return o;
}
```
There are two kinds of threads -- consumer threads \( C1, C2, \ldots \), that remove characters from \( buf \), and producer threads \( P1, P2, \ldots \), that add characters to \( buf \). For our purposes, \( buf.size() \) returns the number of characters in the buffer.

The buffer \( buf \) is initially empty.

1. Consumer \( C1 \) enters the synchronized block for the \( get \) method

2. \( buf.size() == 0 \) is true

3. \( wait() \) is executed, placing \( C1 \) on the lock’s \( wait \) queue

Show the status of \( buf \), lock’s \( wait \) queue and lock’s \( blocked \) queue.
1. Consumer 2 (C2) is just about to enter the synchronized block for the get method, but has not acquired the lock.

2. Producer P1 enters the synchronized method put, acquires the lock, places the character “c” into buf, and calls notify().

3. C1 is woken up by the notify and must reacquire the lock before proceeding. Thus both C1 and C2 are competing for the lock.

Show the status of buf, lock’s wait queue and lock’s blocked queue.
1. One of $C_1$ and $C_2$ is non-deterministically chosen to get the lock. Let’s say $C_2$ gets the lock. It gets to enter the method since $C_1$ is awake and it is put on the *blocked queue*, not back on the *wait queue*.

2. $C_2$ gets the character and releases the lock which is then acquired by $C_1$.

**Is there a character in buf for $C_1$ to get?** No character is available in the buffer.

**What will happen in the program as written?** $C_1$ will go back onto the wait queue to wait for another character to be put into the buffer and to be woken up by a *notify*.

**What would have happened if the while loop was not in the get() method?** $C_1$ would have tried to get a character from the empty buffer, leading to an error.
Let’s look at a scenario that shows the need for `notifyAll` instead of `notify` in the code.
To make this easy, assume a buffer size of 1. Producer and consumer threads are named as before. \( \text{buf} \) is initially empty.

1. \( P1 \) puts a “c” into the buffer.

2. \( P2 \) attempts a \textit{put}, checks the while loop and performs a \textit{wait( )}

3. \( P3 \) attempts a put, checks the \textit{while} loop and performs a \textit{wait( )}

Show the status of \( \text{buf} \), lock’s \textit{wait queue} and lock’s \textit{blocked queue}.
4. The following happen at time step 4:

a. \(C1\) attempt to get 1 character and enters the \textit{get} method;

b. \(C2\) attempts to get 1 character but blocks on entry to the \textit{get} method;

c. \(C3\) attempts to get 1 character but blocks on entry to the \textit{get} method;

Show the status of \textit{buf}, lock’s \textit{wait queue} and lock’s \textit{blocked queue}.
5. The following happen at time step 5.

a. $C_1$ is executing the `get` method, gets the character, calls `notify` and exits the method (releasing the lock and giving $C_2$ and $C_3$ a chance to acquire it);

b. The `notify` wakes up $P_2$

c. BUT, $C_2$ enters the method before $P_2$ can ($P_2$ must reacquire the lock), so $P_2$ blocks on entry to the put method;

d. $C_2$ checks the wait loop, sees there are no more characters in the buffer and so it waits (releasing the lock in the process)

e. $C_3$ enters the method after $C_2$, but before $P_2$, checks the wait loop, sees there are no more characters in the buffer, and so it waits

Show the status of $buf$, lock’s `wait queue` and lock’s `blocked queue`. 
6. The following happen at time step 6.
   
a. Now $P3$, $C2$ and $C3$ are all waiting!
   
b. $P2$ acquires the lock, puts a “$d$” in the buffer, calls \textit{notify} and exits the method
   
Show the status of \textit{buf}, lock’s \textit{wait queue} and lock’s \textit{blocked queue}.

The wait queue is shown before any threads are woken up. See the next page (time step 7) for what is woken up.
7. The following happens at time step 7.

a. P2’s notification wakes up P3 (any thread can be woken up)

b. P3 checks the wait loop condition. There is already a character ("d") in the buffer and so it waits.

Show the status of buf, lock’s wait queue and lock’s blocked queue.

Is it possible for any thread to be woken up by another notify? No.

What would have happened if in 6b a notifyAll() was called? All of C2, P3 and C3 would be woken up and compete for the lock. P3 would get it and C2 and C3 would go onto the blocked queue. P3 would check the while condition and go onto the wait queue, releasing the lock. This would let one of C2 or C3 wake up, take a character, and notify P3. Eventually everyone would finish.
The correct code. Always use `notifyAll` unless there is a good reason not to.

```java
public synchronized void put(Object o) {
    while (buf.size()==MAX_SIZE) {
        wait(); // called if the buffer is full (try/catch removed
        // for brevity)
    }
    buf.add(o);
    notifyAll(); // called in case any getters or putters waiting
}

public synchronized Object get() {
    // Y: this is where C2 tries to acquire the lock (i.e. at the
    // beginning of the method)
    while (buf.size()==0) {
        wait(); // called if the buffer is empty (try/catch removed
        // for brevity)
        // X: this is where C1 tries to re-acquire the lock (see below)
    }
    Object o = buf.remove(0);
    notifyAll(); // called in case any getters or putters waiting
    return o;
}
```